Embedded Linux Size BoF

Michael Opdenacker
michael.opdenacker@bootlin.com

Michael Opdenacker

- Founder and Embedded Linux engineer at ~~Free Electrons~~ → Bootlin
  - Embedded Linux **expertise**
  - **Development**, consulting and training
  - Strong open-source focus
- Long time interest in embedded Linux boot time, and one of its prerequisites: small system size.
- Living in **Orange**, France.

Why reduce kernel and system size?

`https://tiny.wiki.kernel.org/use_cases`

- Run on very small systems (IoT)
- Run on old machines
- Cut costs
- Improve flash lifetime
- Boot faster (for example on FPGAs)
- Reduce power consumption. Even conceivable to run the whole system in CPU internal RAM or cache (DRAM is power hungry and needs refreshing)
- Cloud workloads: optimize instances for size, boot time, network bandwitdh.
- Spare as much RAM as possible for applications and maximizing performance.
- Security: reduce the attack surface
- Run Linux as a bootloader
- Other reasons?

Compiler optimizations

- Using a recent compiler
  Compiling the kernel with gcc 6.3 vs 4.7: only 0.8% smaller size!
- Compiling with gcc LTO
  Compiling `oggenc.c` with `-Os -flto` instead of `-Os`:
  only -2.6% (arm) and -2.8% (x86_x64)
- Using Clang `-Oz` instead of gcc `-Os`
  Compiling `oggenc.c`: -5.7%
- ARM: compiling with `-mthumb` instead of `-marm`:
  -6.8% with `oggenc`
- **Any further technique you'd like to share?**

Reduce user-space size

- Replace BusyBox by Toybox (less configurable, mature and featureful).
  Can save a few tens of KB.
- Replace glibc or uClibc by musl
  musl vs glibc: 76% size reduction in static BusyBox
- For small static executables, musl also wins vs glibc and uclibc
  7300 bytes (musl) vs 492792 (glibc) in static hello world.
- `sstrip` can be used to shave off an extra KB.
- **Any further technique you'd like to share?**

How to get a small kernel?

- Run `make tinyconfig` (since version 3.18)
- `make tinyconfig` is `make allnoconfig` plus configuration settings to reduce kernel size
- You will also need to add configuration settings to support your hardware and the system features you need.

```
tinyconfig:
        $(Q)$(MAKE) -f $(srctree)/Makefile allnoconfig tiny.config
```

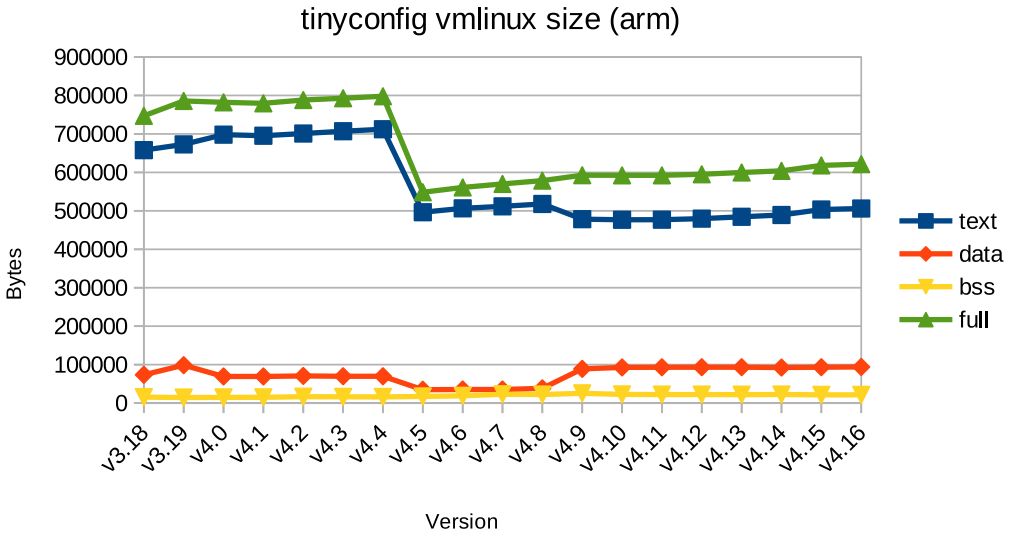
kernel/configs/tiny.config (Linux 4.16)

```
# CONFIG_CC_OPTIMIZE_FOR_PERFORMANCE is not set
CONFIG_CC_OPTIMIZE_FOR_SIZE=y
# CONFIG_KERNEL_GZIP is not set
# CONFIG_KERNEL_BZIP2 is not set
# CONFIG_KERNEL_LZMA is not set
CONFIG_KERNEL_XZ=y
# CONFIG_KERNEL_LZO is not set
# CONFIG_KERNEL_LZ4 is not set
CONFIG_OPTIMIZE_INLINING=y
# CONFIG_SLAB is not set
# CONFIG_SLUB is not set
CONFIG_SLOB=y
CONFIG_CC_STACKPROTECTOR_NONE=y
# CONFIG_CC_STACKPROTECTOR_REGULAR is not set
# CONFIG_CC_STACKPROTECTOR_STRONG is not set
# CONFIG_CC_STACKPROTECTOR_AUTO is not set
```

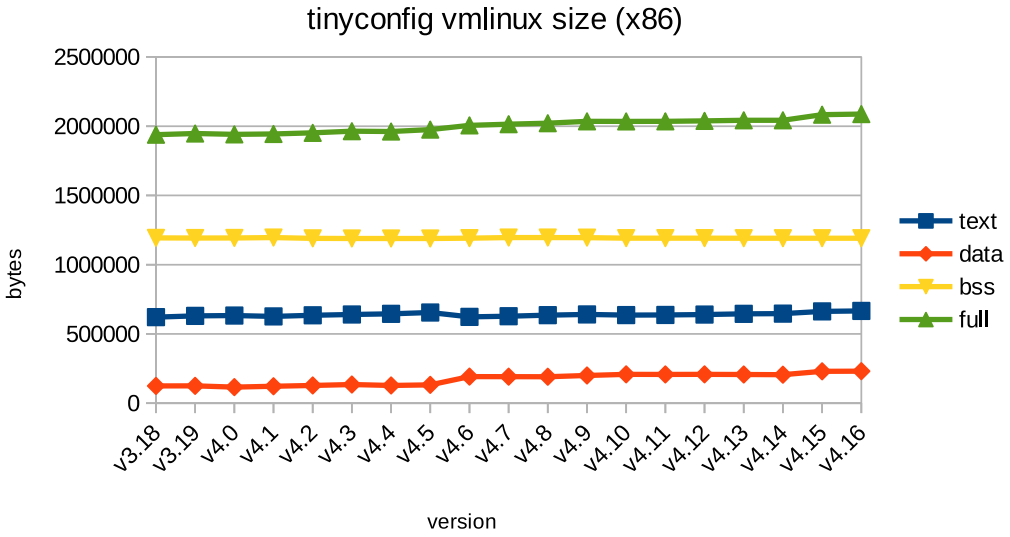
arch/x86/configs/tiny.config

```
CONFIG_NOHIGHMEM=y
# CONFIG_HIGHMEM4G is not set
# CONFIG_HIGHMEM64G is not set
CONFIG_UNWINDER_GUESS=y
# CONFIG_UNWINDER_FRAME_POINTER is not set
```

tinyconfig Linux kernel size (arm)

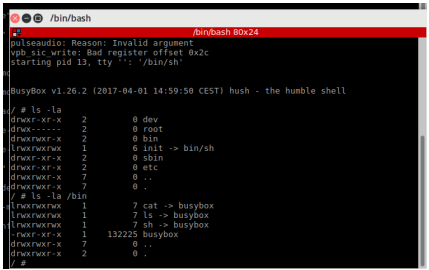


tinyconfig Linux kernel size (x86)



Demo

Mainline Linux 4.15 booting on QEMU
ARM VersatilePB, with 3169 KB of RAM



- `zImage`: 419280 bytes
- text: 1004972 (program code)
- data: 118412 (initialized data)
- bss: 24796 (uninitialized data)
- total: 1148180

Test it by yourself (all in a single line):

```
qemu-system-arm -M versatilepb -nographic -kernel zImage
-initrd initarmfs.cpio.gz -dtb versatile-pb.dtb -m 3169k
```

https://bootlin.com/pub/conferences/2018/elc/opdenacker-size-bof/demo/4.15/

Demo with the Minitty patches

Mainline Linux 4.14-rc4 booting on QEMU ARM VersatilePB, with 2993 KB of RAM

- `zImage`: 393928 bytes (-15 KB vs mainline 4.14-rc4!)
- 128 KB of RAM saved!

Try it by yourself:
`https://bootlin.com/pub/conferences/2017/elce/opdenacker-size-bof/demo/4.14-rc5-minitty/`

Ongoing mainlining efforts

Most (if not all) of the work is currently done by Nicolas Pitre (Linaro). Here are the main ideas:

- Minitty: a minimalistic tty layer for very small systems.
  See `https://lwn.net/Articles/721074/`. Another option is to disable the tty layer completely.
- Nanosched: a lightweight scheduler (little chance to get accepted).
- Instead, proposed to make some scheduling classes optional: sched/deadline and sched/rt.
- Updates to ARM and cramfs for XIP support (merged)

See his latest presentation:
`http://connect.linaro.org/resource/sfo17/sfo17-100/`. Code available on `http://git.linaro.org/people/nicolas.pitre/linux.git`

Reaction from upstream maintainers

Ingo Molnar, June 11th 2017

But you can prove me wrong: show me a Linux kernel for a real device that fits into 32KB of RAM (or even 256 KB) and then I'll consider the cost/benefit equation. Until that happens I consider most forms of additional complexity on the non-hardware dependent side of the kernel a net negative.

Supporting real hardware

To convince upstream maintainers, Nicolas' ultimate goal is to run Linux on the STM32F469NI MCU:

- BGA216 package
- ARM Cortex-M4 core
- 2 Mbytes of Flash
- 324 Kbytes of RAM

Nicolas started to work on the STM32F469 Discovery kit (with 16 MB of SDRAM, already well supported by Linux).

Reducing DT size real hardware

Remove the devices you don't use in the mainline DTS

- Managed to reduce the Versatile PB DTS size from 7965 to 1652 (-6 KB), without breaking my demo
- However, didn't manage to run with less RAM. Need to investigate!

Reducing RAM usage

- Running the kernel and user-space in place (XIP, requires NOR flash). Seems to be available only on ARM
- Reducing kernel defines to reduce the size of kernel structures

```
fs/dcache.c
-#define IN_LOOKUP_SHIFT 10
+#define IN_LOOKUP_SHIFT 5
```

- Investigating the big memory consumption from device tree loading. Reducing RAM usage is easier than reducing code size!

How to help with kernel tinification (1)

- Look for `obj-y` in kernel Makefiles:

```
obj-y     = fork.o exec_domain.o panic.o \
            cpu.o exit.o softirq.o resource.o \
            sysctl.o sysctl_binary.o capability.o ptrace.o user.o \
            signal.o sys.o kmod.o workqueue.o pid.o task_work.o \
            extable.o params.o \
            kthread.o sys_ni.o nsproxy.o \
            notifier.o ksysfs.o cred.o reboot.o \
            async.o range.o smpboot.o ucount.o
```

- What about allowing to compile Linux without ptrace support (14K on arm) or without reboot (9K)?
- Another way is to look at the compile logs and check whether/why everything is needed.

How to help with kernel tinification (2)

- Look for tinification opportunities, looking for the biggest symbols:
  `nm --size-sort vmlinux`
- Look for size regressions with the *Bloat-O-Meter*:

```
> ./scripts/bloat-o-meter vmlinux-4.9 vmlinux-4.10
add/remove: 101/135 grow/shrink: 155/109 up/down: 19517/-19324 (193)
function                                     old     new   delta
page_wait_table                                -    2048   +2048
sys_call_table                                 -    1600   +1600
cpuhp_bp_states                              980    1800    +820
...
```

Projects to follow

- Compiling Linux with LLVM/Clang
  Google (Greg Hackmann and Nick Desaulniers) managed to compile the 4.4 and 4.9 stable kernels, opening the door to size and performance optimizations: `https://lwn.net/Articles/734071/`. Apparently, no progress since Sep. 2017.
- Compiling Linux with gcc LTO
  Some efforts (Andy Kleen, Nicolas Pitre) but not in mainline yet. The patches started in 3.x, but support for recent versions (i.e. 4.15) is still available at `https://github.com/andikleen/linux-misc/`. Read Nicolas Pitre's article on `https://lwn.net/Articles/744507/`

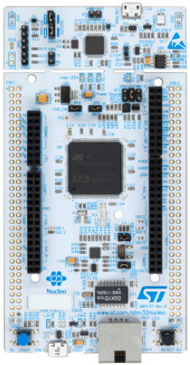
Other ideas

- Resurrect patches from Josh Triplett which didn't get in: `https://git.kernel.org/cgit/linux/kernel/git/josh/linux.git/`
- Simplify the filesystem layer: you don't want things like readahead or page writeback support when you don't have storage. However, that's very difficult to remove!
- Remove kernel features such as ptrace, reboot support, etc.
- Revive single-user (`CONFIG_NON_ROOT`) support: `https://lwn.net/Articles/631853/`
- Modify the kernel binary to remove symbols that were not used during runtime tests (must have full coverage)? At least, can be done without hurting the mainline code! **How to do that?**
- **Other ideas?**

Another hardware platform worth supporting

- Try to support a board with no SDRAM:
  - 512K of on-chip RAM
  - 2M of flash
  - ARM Cortex M7 CPU
  - Cost: 25 USD / 20 EUR

Hoping to have a system with a very good battery life!



STM32 Nucleo F767ZI

Useful resources

- Nicolas Pitre's articles on Linux kernel size
  - Shrinking the kernel with link-time garbage collection `https://lwn.net/Articles/741494/`
  - Shrinking the kernel with link-time optimization `https://lwn.net/Articles/744507/`
  - Shrinking the kernel with an axe `https://lwn.net/Articles/746780/`
  - Shrinking the kernel with a hammer `https://lwn.net/Articles/748198/`
- My detailed presentation about reducing Linux size (with benchmark details) `https://bootlin.com/pub/conferences/2017/jdll/opdenacker-embedded-linux-in-less-than-4mb-of-ram/`
- Ideas ideas and projects which would be worth reviving `http://elinux.org/Kernel_Size_Reduction_Work` (needs updating!)

Questions?

Michael Opdenacker

michael.opdenacker@bootlin.com

Support our crowdfunding campaign to develop
an upstream Linux kernel driver for Allwinner VPU
https://bootlin.com/blog/allwinner-vpu-crowdfunding/